Improved discrete cuckoo search for the resource-constrained project scheduling problem ☆

Kirils Bibiks, Yim Fun Hu, Jian-Ping Li, Prashant Pillai, Aleister Smith*
Faculty of Engineering and Informatic, University of Bradford, Bradford, BD7 1DP, United Kingdom

Abstract

An Improved Discrete Cuckoo Search (IDCS) is proposed in this paper to solve resource-constrained project scheduling problems (RCPSPs). The original Cuckoo Search (CS) was inspired by the breeding behaviour of some cuckoo species and was designed specifically for application in continuous optimisation problems, in which the algorithm had been demonstrated to be effective. The proposed IDCS aims to improve the original CS for solving discrete scheduling problems by reinterpreting its key elements: solution representation scheme, Lévy flight and solution improvement operators. An event list solution representation scheme has been used to present projects and a novel event movement and an event recombination operator has been developed to ensure better quality of received results and improve the efficiency of the algorithm. Numerical results have demonstrated that the proposed IDCS can achieve a competitive level of performance compared to other state-of-the-art metaheuristics in solving a set of benchmark instances from a well-known PSPLIB library, especially in solving complex benchmark instances.

Keywords: Scheduling; Resource-constrained project scheduling problem; Cuckoo search; Metaheuristics; Combinatorial optimisation

1 Introduction

Scheduling has been an active research topic in optimisation for many years. In literature, a variety of scheduling problems have been proposed (Tritschler *et al.*, 2017; Hartmann and Briskorn, 2010; Zhou and Zhong, 2007; and Hsu *et al*.,2004). Despite the variety, the majority of them can be classified as variations of Resource-Constrained Project Scheduling Problems (RCPSPs). The objective of an original RCPSP is to find a schedule for the minimal duration of a given project which consists of a set of activities with known deterministic durations and a set of resources with limited capacities. Blazewicz *et al*. (1983) described RCPSPs as a generalisation of a classical job-shop scheduling problem (Chen and Quan, 2008) and showed that they belongs to a class of NP-hard combinatorial optimisation problems.

In the last decades, RCPSPs have received a lot of attention because of the relative generality and numerous practical applications (Kolisch and Padman, 2001; and Herroelen *et al.,* 1998). As the result of this, several methods have been proposed for solving them. Kolisch and Hartmann (1999b) divided all methods into two categories: exact methods and heuristics. Further, heuristics was additionally divided into priority rule-based methods and metaheuristics.

Tormos and Lova (2001) in their research tested several popular exact solution procedures and heuristics for RCPSPs. Performances of the selected algorithms were measured using the setup proposed by Kolisch and Sprecher (1997). The most competitive exact algorithms were the ones of Brucker *et al.* (1998), Mingozzi *et al.* (1998), Specher (2000) and Demeulemeester and Herroelen (1992). Nevertheless, even though these exact algorithms demonstrated good performances, in a satisfactory manner they were only capable of solving small-scale instances of problems with up to 60 activities.

Some evolutionary computations such as genetic algorithm (GA) (Alcaraz and Maroto, 2001), ant colony optimisation (ACO) (Merkle *et al*., 2002), simulated annealing (SA) (Bouleimen and Lecocq, 2003), Bee Algorithm (BA) (Ziarati *et al.*, 2011) and hybrid algorithms (Myszkowski *et al.*, 2018) have been

extensively applied to solve RCPSPs. Husbands *et al.* (1996) demonstrated that stochastic optimization techniques, such GAs, can find better solutions and illustrated the resemblance between scheduling and sequence-based problems. Hartmann (1998) suggested that every gene of a chromosome was a delivery rule. Alcaraz and Maroto (2001) proposed an improved GA for which a new representation of a solution and advanced crossover technique were introduced. Hartmann (2002) presented a self-adapting GA which is capable of adapting to the problems' instances by learning which decoding procedures are more successful. Boctor (1996) was one of the first to successfully apply a SA to solve RCPSPs with good performances. Bouleimen and Lecocq (2003) proposed a SA in which the conventional search scheme was replaced by a new design in which the specificity of the solution space was taken into account. Palpant *et al*. (2004) presented a local search strategy in which a subpart of the current solution was fixed while the other part was formulated as a sub-problem solved by a heuristic or an exact method. Merkle *et al*. (2002) suggested that one of the first uses of ACO was using the summation of the values in the pheromone set for this problem. Kochetov and Stolyar (2003) proposed an evolutionary algorithm that combined a GA, a path relinking strategy and a tabu search. Valls *et al*. (2005) proposed a simple technique called double justification and showed how it can be applied in an optimization technique to improve the quality of a solution without requiring more computing time. They also presented a hybrid genetic algorithm with a peak crossover operator (Valls *et al*., 2002). Kolisch and Hartmann (2006) showed that the aforementioned algorithm could find the best results amongst all other reviewed state-of-the-art heuristics. Ziarati *et al.* (2011) applied three different Bee Algorithms to solve RCPSPs and investigated their performances. Myszkowski *et al.* (2018) presented a hybrid Differential Evolution and Greedy Algorithm (DEGA) to solve Multi-skill RCPSPs and demonstrated that DEGA is robust and effective in solve their 28 instances.

A comprehensive survey done by Hartmann and Kolisch (2000) and its updated version (Kolisch and Hartmann, 2006) provided a classification and performance evaluation of different heuristics for RCPSPs. Performances of some algorithms have been compared when solving different instance sets of the Project Scheduling Problem Library (PSPLIB) generated by ProGen (Kolisch *et al*.,1995). Their results demonstrated that metaheuristic methods outperform heuristic methods. For 26 methods sorted with respect to the performance of evaluating 1000, 5000 and 50,000 schedules, the best methods for J30, J60 and J120 sets are all metaheuristic methods.

In recent years, research on metaheuristics has made significant progress, especially with the arrival of nature-inspired and population-based evolutionary computation. A new nature-inspired metaheuristic method called Cuckoo Search (CS) has been developed by Yang and Deb (2009). Yang and Deb (2010) demonstrated that CS is a very efficient algorithm for finding the global optima with high success rates and that in some cases it is superior to both PSO and GA in terms of efficiency and success rate. CS has managed to attract attention of many researchers from different application fields and domains (Nguyen *et al*., 2016; Teymourian *et al*., 2016; Sekhar and Mohanty, 2016 and Elazim and Ali, 2016). Nevertheless, until recently CS has only had limited amount of applications in optimisation problems with discrete domain. One of the first works that attempted to solve discrete optimisation problem using CS was presented by Ouaarab *et al.* (2013). In their work, the authors used CS to solve the traveling salesman problem. Maghsoudlou *et al.* (2017) developed three cuckoo-search-based multi-objective mechanisms based on non-dominances sorting genetic algorithm, particle swarm optimization and invasive weeds algorithm to solve a real version of multi-skill RCPSP. Further, CS has also found a recent and significant application to the NP-hard annual crop-planning problem (Chetty and Adewumi, 2013).

This paper provides an improved version of a Discrete CS (DCS) algorithm (Bibiks *et al*., 2015) for solving RCPSPs. Previous implementation of the DCS by Bibiks *et al*. (2015) had showed competitive performance against other non-hybrid evolutionary computation. The new version of the DCS, presented in this paper, introduces several changes in the original paradigm: the inclusion of a new category of cuckoos, the use of a novel solution representation scheme specific for the RCPSPs, and a novel local search and event recombination operators.

The rest of the paper is organised as follows. The RCPSP is formally presented in Section 2. The basic CS

and its improved version are outlined in Section 3. Section 4 presents and discusses the Improved Discrete Cuckoo Search (IDCS) algorithm. The experimental results are provided and discussed in Section 5. Finally, Section 6 concludes the paper.

2 Problem Description

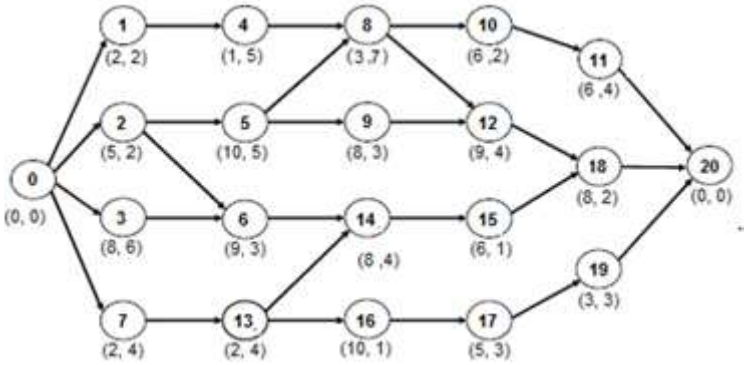
Activities in a RCPSP are defined as a finite set $\boldsymbol{V} = \{0, 1, \ldots, n, n+1\}$, where n is the number of activities in a project. Activities 0 and $n+1$ are referred to as dummy activities, which denote the start point and the end point of a project respectively. Each activity i $(1 \leqslant i \leqslant n)$ has a duration time $d_j \geq 0$. The duration of dummy activities is set to $d_0 = d_{n+1} = 0$. Activities are represented by an activity-on-node (AON) graph (also called project network) $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ (Zhou and Chen, 2002), where $\boldsymbol{V}$, the active set, is the set of vertices and $\boldsymbol{E}$ is set of arcs representing precedence relations of activities. Notation $(i, j) \in \boldsymbol{E}$ means that activity j can be started only after activity i has been completed. An example of a project network can be seen in Fig. 1(a). A project /schedule is defined as a connection of related activities.

The availabilities of renewable resources are defined by a finite set $\boldsymbol{R} = \{R_1, R_2, \ldots, R_K\}$. The amount of Resource k required for the execution of activity i is denoted by r_{ik}.

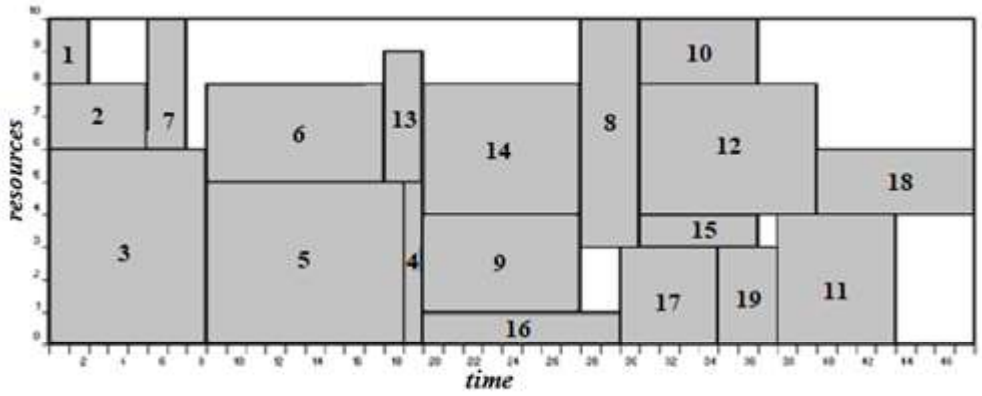
The starting times of activities are represented by a schedule $\boldsymbol{S} = \{s_0, s_1, \ldots s_j, \ldots, s_n, s_{n+1}\}$, where s_j is the starting time of activity j. s_0 is the start of a project and is always assumed to be 0 and the finishing time of activity j is denoted as f_j. If an activity is not included in a schedule, its starting time is set to a negative number. The relationship between the starting time and the finishing time is defined as:

$$s_j + d_j = f_j \tag{1}$$

The total duration of a project, or its makespan, will be equal to the finishing time of the last activity f_{n+1}.



(a) Project Network



(b) Sample schedule

Fig. 1. Project example

Finally, a RCPSP can then be described as finding a non-pre-emptive schedule ***S*** with minimal makespan f_{n+1} subject to resource and precedence constraints, as indicated in (3) to (5) below respectively:

Min: $$f_{n+1} \quad (2)$$

Subject to: $$\sum_{i=1}^{n} r_{ik} \cdot a_i(t) \le R_k \quad \forall k = 1,2,\Lambda\ ,K \text{ at any given time t} \quad (3)$$

$$f_0 = 0 \quad (4)$$

$$s_j - s_i \ge d_i \qquad \forall\ (i,j) \in \mathbf{E} \quad (5)$$

Where $a_i(t)$ is the status of Activity *i* at Time *t* and defined as

$$a_i(t) = \begin{cases} 1 & s_i \le t < f_i \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

Equation 3 shows that the resource demoand of activities at any time (*t*), the sum of all resource demands of activities that are active at Time *t*, must be lower or equal to resoursce avaiablity, while Equation 5 is the precedence relation restriction.

A solution is defined as a schedule / project which consists of a series of activities. The decision variable of the proposed problem is a schedule S and descried as the order of activities.

Fig. 1 displays an example taken from Debels and Vanhoucke (2007) comprising of 19 non-dummy activities and a single resource with a capacity of 10 units. In Fig. 1(a) a project network is displayed. Under each node a duration and resource request of the corresponding activity are provided. In Fig. 1(b) a feasible

schedule of makespan (47) is visualized in the form of a chart. The horizontal axis of the chart shows the time when each activity is going to be executed, while the vertical axis shows the amount of resources that will be taken. Each block on the chart corresponds to an activity from the sample project. In subsequent sections of the paper this example is going to be used to illustrate some operations of the proposed algorithm.

3 Cuckoo Search

CS was inspired from a breeding behaviour of cuckoo species. In the original version of CS explained by Yang and Deb (2010) cuckoos are illustrated as basic search agents, where each egg they dump serves as a candidate solution for a problem. The basic CS is based on three principles:

1. Each turn a cuckoo only lays one egg which is then placed in a randomly selected nest;
2. Nests with the fittest eggs will survive in the next generation;
3. The total number of host nests is constant. At the end of each turn, a fraction $p_a \in [0, 1]$ of worst nests is abandoned and replaced with newly generated ones.

Cuckoo Search

Initialise a population P of m host nests $\mathbf{x}_i$, $P_m = (x_1, x_{1,} \ldots, x_m)$

For all $\mathbf{x}_i$ **do**
  Calculate fitness $F_i = f(\mathbf{x}_i)$
End for
While (ObjectiveEvaluationNumber < MaxEvaluationNumber)
  Get a cuckoo randomly and generate a new solution ($\mathbf{x}_j$) by Lévy Flights
  Evaludate fitness $F_j = f(\mathbf{x}_j)$
  Choose random individual $\mathbf{x}_i$ from the population P_m
  If ($F_j > F_i$) **then**
    Replace $\mathbf{x}_i$ with $\mathbf{x}_j$
  End if
  Abandon a fraction p_a of individuals with worst fitness which will be replaced by new randomly generated individuals.
End while
Find the fittest individual

Fig. 2. Cuckoo search pseudo-code

The final principle can be understood as follows. If the abandonment rate parameter p_a is set to 0.2, then at the end of each iteration 20% of worst nests will be replaced with new randomly-generated ones.

The steps of a basic CS algorithm are outlined in Fig. 2. As can be noted from the above pseudo-code, an important aspect of the CS is the use of Lévy flight for both local and global searching. The Lévy flight process, which has previously been used in other search heuristics (Pavlyukevich, 2007), is a random walk that is characterised by a series of instantaneous moves chosen from a probability density function which has a power law tail. This process represents the optimum random search pattern and is frequently found in nature (Viswanathan, 2008).

When generating a new individual, a Lévy flight is performed. If the objective function (i.e. fitness) of the new individual is better than the objective function of another randomly selected one, the new individual replaces it. The scale of this random search is controlled by multiplying the generated Lévy flight by a step size α:

$$x_i^{(t+1)} = x_i^t + \alpha \oplus Levy(\lambda) \quad (7)$$

where Lévy flight represents a random walk where step lengths are distributed corresponding to a heavy-tailed probability distribution.

4 Improved Discrete Cuckoo Search

The original CS was developed with the intention for applications in continuous optimisation problems. This algorithm can, therefore, not be used directly to solve problems in a discrete domain (i.e. RCPSP). In this paper, the ideas of the CS are extended to a discrete domain through the change of the algorithm's original paradigm.

One of the major goals of extending CS to solve the RCPSP is to retain its key advantages and incorporate them into the discrete version of algorithm. The adaptation of the CS to RCPSP primarily emphasises the reinterpretation of its key elements and operators. The fundamental basis of CS can be structured and presented by two key elements: solution representation scheme and exploration and exploitation of the solutions search space via Lévy flight.

4.1 Solution Representation Scheme

When solving optimisation problems, one of the most important factors that influences the performance of an algorithm is the efficiency of a solution representation scheme. For RCPSPs it is more convenient to operate on an encoded solution rather than its direct form. Such approach allows simultaneous consideration of both precedence and resource constraints when creating new individuals.

Various solution representation schemes exist for RCPSPs in the literature. Kolisch and Hartmann (1999a) produced a comprehensive review of the most popular ones and their respective operators. Among all reviewed solution representation schemes, two did standout the most: activity-list (AL) and random-key (RK).

The AL representation scheme represents a vector (i.e. list) comprised of n activities. The index for each of the AL's elements depicts the order in which an activity is going to be scheduled; hence, activities in the AL are scheduled in the same order as they appear. The RK, on the other hand, encodes a solution as a vector of n numbers where the i^{th} number relates to the i^{th} activity. RK is transformed into a schedule by successively scheduling activities with the highest random keys (random numbers). Based on computational experiments conducted on sets of benchmark instances from PSPLIB, Kolisch and Hartmann (2006) concluded that AL is more efficient than the RK and algorithms that operate on this representation scheme tend to produce better results. In the performance evaluation of more than 20 heuristics for the RCPSP, the top 8 algorithms operated on AL representation scheme.

On the other hand, Moumene and Ferland (2008) proposed to decode solutions by using Activity Set List (ASL), which represents an ordered subset list of different non-empty activities. Each subset consists of a group of activities which share common characteristics, such as predecessors and successors; therefore, by using an ASL, the search space is significantly reduced. Paraskevopoulos *et al*. (2012) continued to improve ASL representation and proposed a new solution representation scheme, called Event List (EL). Similar to the ASL, Paraskevopoulos *et al*. (2012) grouped activities with identical characteristics (i.e. starting times) into sets, called events. In the EL representation, no adjustment and repairing mechanisms were used. The evolution operation was done on events, not activities. This makes the search more efficient.

Fig. 3 illustrates AL and EL representations of the project shown in Fig. 1. The numbers in the boxes represent the activity index numbers, while the numbers below the boxes show their corresponding starting times. Activities with the same starting times are put together as an event in an EL representation. Therefore, a schedule (or solution) is a set of events ordered by their starting times. The number of activities in an event is called the order of the event.

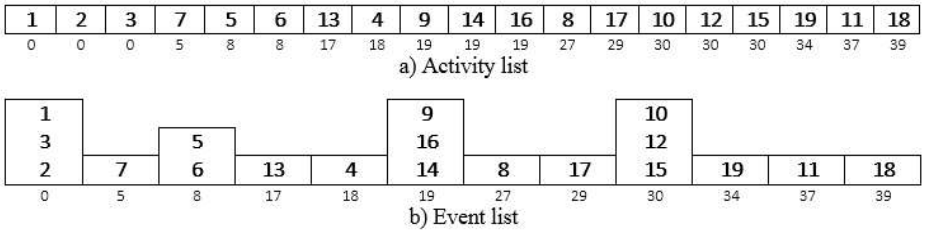


Fig. 3. Event list representation example

4.2 Improved Lévy Flight

4.2.1 Lévy Flight

Yang and Deb (2009) demonstrated that Lévy Flights could improve the quality of solutions for some optimisation problems.

In a nutshell, Lévy flights can be understood as intensive searches in local areas around performed in small steps around the current solution, followed by occasional large jumps. A step represents a distance in the search space from the current position to a new solution. In this paper, the length of a small step is defined as the number of event movements that will be performed on a current solution, whereas a large jump is implemented with event combination. In order to improve the search quality, the length of a step will be dependent on the value generated from the Lévy distribution.

Several implementations of algorithms for generating Lévy distribution values exist in the literature. Leccardi (2005) compared different approaches to generating Lévy values and demonstrated that the algorithm by Mantegna (1994) was the most effective method. Mantegna (1994) algorithm produces a random noise according to a symmetric Lévy stable distribution, which is ideal for Lévy flights. In this paper, this method is applied to determine the length of step.

In the Mantegna (1994) algorithm, the step length was calculated as:

$$Levy(\lambda) = \frac{u}{|v|^{\frac{1}{\lambda}}} \quad (8)$$

where u and v are drawn from normal distributions defined by:

$$\begin{aligned} u &\sim N(0, \sigma_u^2) \\ v &\sim N(0, \sigma_v^2) \end{aligned} \quad (9)$$

$$\sigma_u = \frac{\Gamma(1+\lambda) * \sin(\frac{\pi\lambda}{2})}{\Gamma(\frac{1+\lambda}{2}) * \lambda * 2^{\frac{\lambda-1}{2}}}, \ \sigma_v = 1 \quad (10)$$

where the distribution parameter $\lambda \in [0.3, 1.99]$ and Γ denotes Gamma function.

4.2.2 Lévy Flight Operation Process

The amount of Lévy Flight steps is associated with the value generated using Mantegna (1994) algorithm

in the interval between 0 and 1. In the proposed algorithm, the following strategy is used to improve the efficiency of the algorithm: the steps are determined in accordance with the Lévy distribution value. The relationship between the Lévy value and the corresponding operation are defined (Yang 2013) as:

1. Lévy in [0, i] – perform one small step
2. Lévy in [(k-1)*i, k*i] – perform k amount of small steps
3. Lévy in [k*i, 1] – perform large jump

where the value of i in this process is i = (1/(s+1)), s is a configurable parameter representing the maximum number of steps and k is in [2, .., s]. For example, if s = 3, then the whole interval will be divided in 4 parts:

1. Lévy in [0, 0.25] – 1 step
2. Lévy in [0.25. 0.5] – 2 steps
3. Lévy in [0.5, 0.75] – 3 steps
4. Lévy in [0.75, 1] – large jump

For continuous optimisation problems, the step size of a movement represents a distance between two solutions in the search space. In this paper, a CS was applied for solving combinatorial optimisation problems, therefore a small step size is defined as the number of events that will be relocated by an event movement operator, whereas a big step is represented by an event recombination operator. The new Lévy flights is the combination of EL evolution operations: event movement and event combination, which are controlled by the generated Lévy value. An event combination operator was developed based on the concept of crossover used in Genetic Algorithm. Our experiences showed that the event combination operation has a large influence on the quality of solutions.

The pseudo-code of the improved Lévy Flights is shown in Fig. 4.

Improved Lévy Flights

For a given cuckoo $\mathbf{x_i}$ and max amount of steps **s;**
Generate Lévy number by using Equation (5);
If (Lévy <= 1 - (1/(**s**+1))) **then**
  Flight step= Lévy/0.25;
  Implement the event movement with the flight steps described in Section 4.2.3;
Else
  Implement the event recombination described in Section 4.2.4;
End if
Apply SGS for objective evaluation

Fig. 4. Improved Lévy Flight

4.2.3 Event Movement

The original idea of the event movement operator, applied in this paper, was proposed by Paraskevopoulos *et al*. (2012).

As was stated previously, an event in the EL represents a set of activities with the same starting times. Generally, these activities may share the same project characteristics, such as the same predecessors and/or successors. If so, they can be considered as one entity. This trait served as the main influence for the creation of event movement operator. The algorithm of event movement is summarised in the pseudo-code in Fig. 5.

Event Movement

Initialise event list E of size k, $E_k = (e_1, e_2, \ldots, e_k)$

Randomly pick an event $e_i = (a_1, a_2, \ldots, a_n)$ from E_k
For all $\mathbf{a}_j$ in e_i **do**

Find allowable range of positions for relocation for a_j
Relocate a_j to new position
End for

Fig. 5. Pseudo-codes of Event movements

When the event movement is performed on an EL, a number of events are randomly picked for relocation. The number of picked events depends on the amount of small steps that are calculated based on the Lévy value. Possible positions for a relocation of selected events are calculated according to their precedence relations between activities: positions of the latest starting predecessor and the earliest starting successor. Each activity in the selected event is moved independently from each other to random positions within their allowable possible positions. These activities might be added to existing events or form a new event if no suitable event exists.

Once relocations of the selected events are completed, a schedule generation scheme (SGS) is applied to produce a new schedule and its makespan is calculated. When SGS is applied, all the activities will be rescheduled independently and the starting times will change.

Event movement is a local search. Fig. 6 demonstrates an example schedule that has resulted from the event move. The sample project from Fig. 1 is used as the initial solution. The whole process of event moves goes as follows:

- Step 1: an event is randomly selected. In this case, it is assumed that the highlighted event shown in Fig. 6a is picked. This event consists of Activities *9*, *14* and *16.*
- Step 2: The selected event is removed from the EL and the activities in the selected event are randomly inserted into the schedule: Activity *9* is inserted between Activities *8* and *10*, while Activities *14* and *16* were inserted between Activities *10* and *12*.
- Step 3: The above process will be repeated based on the step size.
- Step 4: The makespan is evaluated via the application of the SGS.

The final EL is given in Fig. 6c, while its schedule is presented in Fig. 6d. As can be noted, after activities of the picked event have been rescheduled to different positions, the makespan of the schedule is reduced from 47 to 45.

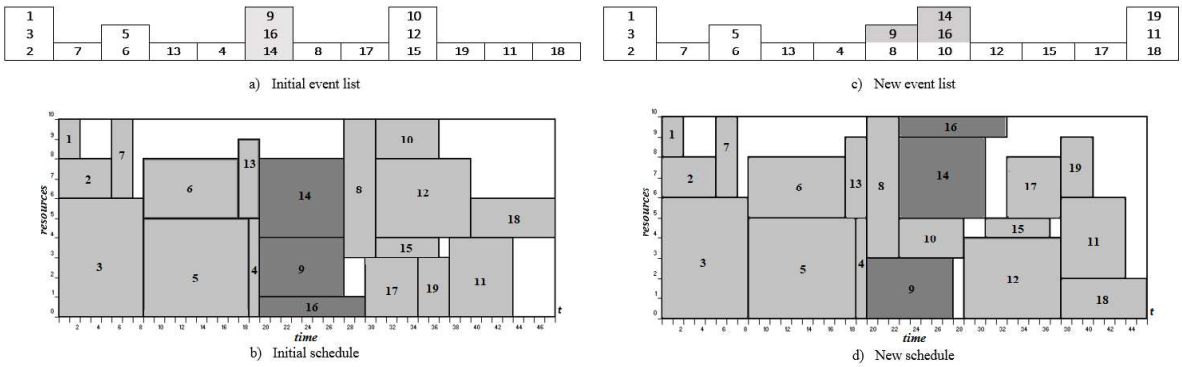


Fig. 6. Event movement example

4.2.4 Event Recombination

The proposed event recombination operator operates on the EL representation. Instead of using random activity chains of solution parts to be recombined (i.e. the approach followed in AL-based methodologies), the proposed operator treats events as the solutions' elements for recombination. Given two solutions, Parent 1 and Parent 2, the event recombination operator generates an offspring in such a way that events with the highest amount of activities are inherited from Parent 1, whereas the positions and order of the remaining activities are determined by Parent 2

The full sequence of steps of this operator is outlined in the pseudo-code in Fig. 7.

Event Recombination

Initialise 2 event lists, E_x and E_y, each consisting of n activities

Sort all events e_k in E_x by size in descending order
Start picking largest e_k from E_x until total amount of picked activities is >= n/2
Pick remaining activities from E_y in the same order as they appear
Form new E_z from picked events

Fig. 7. Pseudo-code of Event recombination

The event recombination process begins by ordering events in Parent 1 in descending order by their sizes. Then the events are picked one by one from the largest to the smallest until the number of activities that comprise the picked events is 50% from the total number of activities. For Parent 1, three events are selected for the crossover, each of which consists of three activities. The remaining activities are then taken from Parent 2 in their respective order.

The procedure shown in Fig. 8 builds an offspring and assigns activities to their respective positions one by one in a growing position order. At each moment, an activity is a candidate if it does not belong to any existing event and has not been assigned, while all its predecessors have already been added. It is worth noting that during its creation, the offspring is represented in a form of AL. The resulting AL, shown in Fig. 8a, with the application of a serial schedule generation scheme is then converted into EL by scheduling each activity. Finally, a schedule with the makespan of 45 is produced, as demonstrated in Fig. 8b.

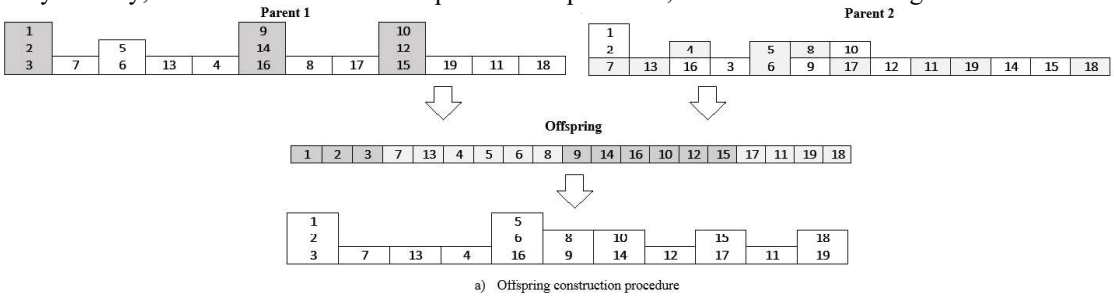


Fig. 8. Event recombination example

4.3 Pseudo-Codes of the developed algorithm

In addition to that, the original paradigm of the algorithm is implemented with a new mechanic which each turn aims to improve a fraction of individuals via local search. In this paper, the implementation of Lévy flights relies on the combination of event movement (local search) and event recombination (global search) operators. The pseudo-code of the IDCS can be seen in Fig. 9.

Improved Discrete Cuckoo Search

Initialise a population P of m host nests $\mathbf{x}_i$, $P_m = (x_1, x_{1,}, \ldots, x_m)$
For all $\mathbf{x}_i$ **do**
  Calculate fitness $F_i = f(\mathbf{x}_i)$
End for
While (ObjectiveEvaluationNumber < MaxEvaluationNumber)
  Smart search: randomly select a fraction (p_c) of cuckoo and move them to new positions by Lévy Flight (Section 4.2)
  Get a best cuckoo and generate a new solution ($\mathbf{x}_j$) by Lévy Flight (Section 4.2)
  Evaluate fitness $F_j = f(\mathbf{x}_j)$
  Choose random individual $\mathbf{x}_i$ in a nest (say j)
  If ($F_j > F_i$) **then**
    Replace $\mathbf{x}_i$ with $\mathbf{x}_j$
  End if
  Abandon a fraction p_a of individuals with worst fitness which will be replaced by new randomly generated individuals.
End while
Find the fittest individual

Fig. 9. Improved discrete cuckoo search pseudo-code

The improved Lévy flight operator consists of event movements and event recombination. This means that there are more than operations on one schedule. In the proposed algorithm, a schedule can only been evaluated only after all the operations have been completed, and therefore only one fitness evaluation is accounted.

The proposed IDCS does not have an explicit local search operator. However, the proposed improved Lévy Flight is the combination of local search and global search. There are two operators in the Lévy flight event movement and event recombination. If the flight step size is small (such as 1), it performs a local search. If two parents are close each other, the event recombination is a local search.

In the smart search, a fraction (p_c) of cuckoos are randomly selected from the current populations. Each cuckoo will be placed to a new position by applying the improved Lévy flight to explore solutions in the search space.

The best cuckoo will be moved to a new place by using the improved Lévy flight. The new individual will be compared with a local cuckoo. If the new cuckoo is better than the selected local cuckoo, the local cuckoo will be replaced with the new generated cuckoo. This operation aims to improve the global solutions.

However, the algorithm may stuck in some local solutions. In order to improve the capability of global search,

there is an abandoning operation. In each generation, a proportion (p_a) of *cuckoo*s will be abandoned and some new cuckoos will be randomly generated.

5 Computational Performance

To evaluate the performance of the IDCS, various numerical experiments are conducted on sets of benchmark instances from project scheduling problems library (PSPLIB) (Sprecher, Kolisch, & Drexl, 1995) designed specifically for testing RCPSP methodologies. PSPLIB contains instances of scheduling problems with varying difficulty, which are grouped into sets in accordance with the amount of activities each project contains. The following sets are available:

- J30: 480 instances of scheduling problems, each consisting of 30 activities and 4 resource types;
- J60: 480 instances of scheduling problems, each consisting of 60 activities and 4 resource types;
- J120: 600 instances of scheduling problems, each consisting of 120 activities and 4 resource types.

In order to assess the performance of the algorithm after running each of the benchmark instances, a deviation from the optima is calculated. For J30 instances, a deviation is calculated with respect to optimal solutions, while for the J60 and J120 instances, a deviation is calculated with respect to the length of the critical path (CP). CP is obtained by computing the makespan of a project by relaxing the resource constraints of the problem.

5.1 Experimental Setup

Before the performance of the algorithm can be evaluated and compared with others, it is necessary to configure it and find the most appropriate parameter settings. To do this, the *irace* package (Birattari et al. 2010) is utilised in the experimental setup. The *irace* package is an automatic configuration tool to automatically find the best parameter settings of an optimisation algorithm in order to achieve its best performance. By receiving a list of parameters used in an algorithm as inputs, *irace* uses a set of training instances to automatically find the optimal levels for each parameter. This is achieved by searching the parameter search space for good performing algorithm configurations through executing the target algorithm on different instances with different parameter configurations.

In this experimental setup, *irace* is set to use benchmark instances from PSPLIB to tune the algorithm. In this setup benchmark instance from J30, J60, and J120 sets have been utilised for the tuning of the target algorithm:

- J30 set – every tenth instance starting from number 1, 48 instances total;
- J60 set – every tenth instance starting from number 1, 48 instances total;
- J120 set – every tenth instance starting from number 1, 60 instances total.

The stopping criterion for running each of these instances was set to 5000 objective evaluations.

After the algorithm is configured and optimal parameters are identified, its performance can be evaluated and compared against other methodologies. Typically, the performance of the algorithms for the RCPSP are evaluated by running all benchmark instances from J30, J60, and J120 sets from PSPLIB. In order to provide the basis for comparison with other algorithms, Hartmann et al. (2000) suggested to limit the execution of algorithms to the amount of time in which the objective function is evaluated (i. e. the number of generated schedules). The advantage of this stopping criterion is that it is independent of the computer platform. Therefore, all heuristics can be tested using the original implementation and the best configuration. Moreover, such tests are independent of compilers and implementation skills, thus the concept of the algorithm is evaluated, rather than its program code. Hence, in order to evaluate the performance of an algorithm, three sets of experiments have been conducted in which the algorithm will have to run all benchmark instances

from J30, J60, and J120 sets for the three stopping criteria (maximum of 1000, 5000, and 50000 objective function evaluations).

5.2 Parameter Settings

The IDCS has four configurable parameters:

- Population size m
- Abandonment rate p_a
- Max amount of steps s
- Portion of smart cuckoos p_c

In order to find the optimal values for these parameters, a sensitivity analysis has been carried out using an *irace* package. The ranges of parameters' values selected for the analysis are summarized Table 1.

Table 1. IDCS parameter values for sensitivity analysis

Parameter	Values Range
Population size (m)	[10, 200]
Abandonment rate (p_a)	[0, 0.9]
Max amount of steps (s)	[1, 10]
Portion of smart cuckoos (p_c)	[0, 0.9]

As the result of the algorithm fine-tuning, the optimal parameters values identified by using the *irace* package are summarized in **Table 2**.

Table 2. IDCS optimal parameters values

Parameter	Value
Population size (m)	18
Abandonment rate (p_a)	0.7
Max amount of steps (s)	4
Portion of smart cuckoos (p_c)	0.2

5.3 Performance Analysis

During the tuning process, *irace* iteratively updated the sampling models of the parameters to focus on the best regions of the parameter search space. The frequency of the sampling of parameters' values in the regions of the specified parameters' search space for m, p_a, s and p_c is presented in Fig. 10-13 respectively.

As can be observed from the above-presented graphs, the optimal levels of DCS parameters were in the following ranges:

- $m - [10; 20]$
- $p_a - [0.7; 0.9]$
- $p_c - [0.1; 03]$
- $s - [2; 4]$

Due to the high abandonment rate, the overall population is constantly updated with new individuals, hence, there is no need for maintaining high population. By keeping the maximum amount of the step parameter, *s*, values between 2 and 4, the algorithm provides the perfect balance between usage of pairwise interchange and shift operators.

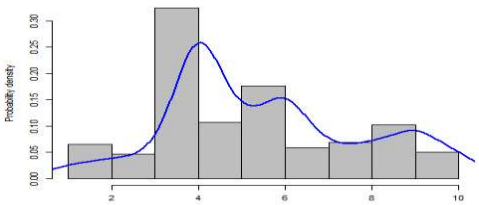


Fig. 10. Max amount of steps sampling frequency

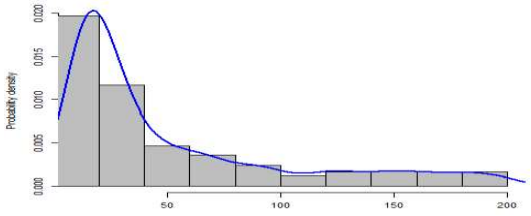


Fig.11. Population size sampling frequency

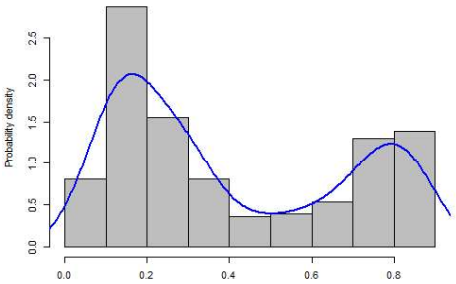


Fig.12. Portion of smart cuckoos sampling frequency

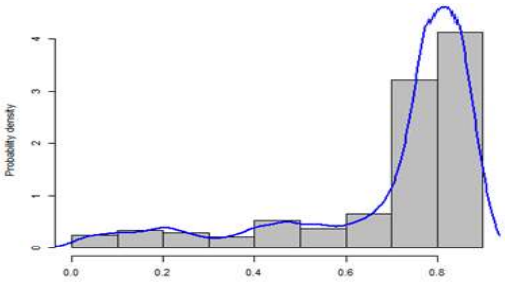


Fig. 13. Abandonment rate sampling frequency

Furthermore, graphs presented in Fig. 14-17 demonstrate how the variance of parameter values affect the overall quality of the final solution. The presented results were obtained by doing four sets of experiments on randomly selected benchmark instance from J120 set. In each of the experiments the value of the parameter under test was continuously incremented, whereas values of other parameters were kept constant at all times.

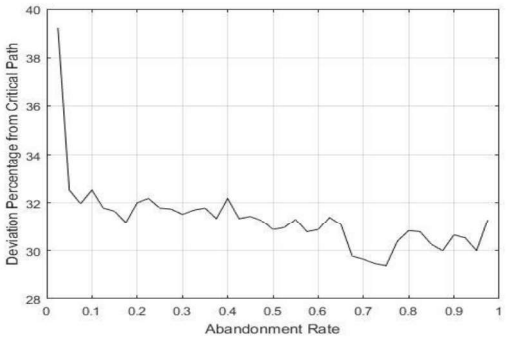


Fig.14. Effect of abandonment rate on the final result

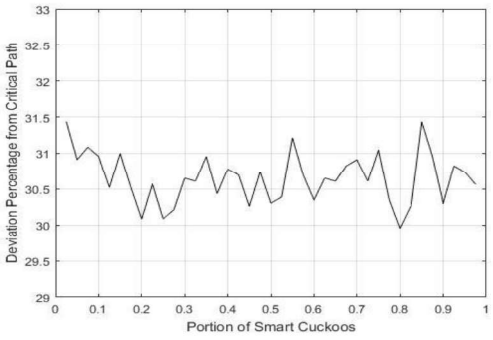


Fig.15. Effect of portion of smart cuckoos on the final result

For each combination of parameters in the above-presented figures, the algorithm ran benchmark instance 50 times and the average value of all received solutions was taken. The stopping condition in all experiments

was set to 1000 objective evaluation. The values of constant parameters were set to the one identified by *irace* package (Table 2).

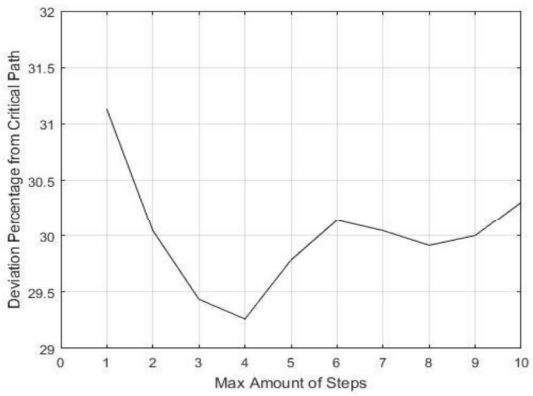


Fig.16. Effect of max amount of steps on the final result

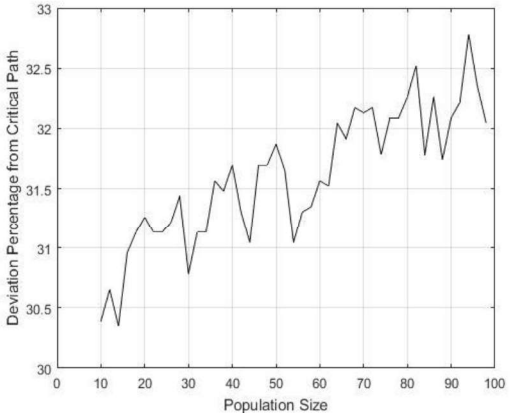


Fig.17. Effect of population size on the final result

5.4 Comparative Analysis

The performance of the proposed IDCS has been studied and compared with other state-of-the-art methodologies taken from the survey on state-of-the-art computational methods for the RCPSP (Kolisch and Hartmann, 2006). The first implementation of the DCS (Bibiks *et al*., 2015) is also included in the comparison list and referenced as *Original DCS*. However, it is worth mentioning that this implementation of the DCS (Bibiks *et al*., 2015) was only used to solve instances from J30 set. Additionally, in order to further examine the effectiveness of the event recombination for the realisation of big step movement, a special version of the proposed algorithm is utilised to conduct numerical experiments. The main difference of this version of the algorithm from the one that is presented in Section 4 is the disuse of big step movement and, as the result of that, the application of only the event move operator for exploration of the search space. This algorithm is referenced as *Special IDCS.*

The computational results of the evaluation of the proposed algorithm are shown in Table 3, 4, and 5 for the instance sets J120, J60, and J30, respectively. The abbreviations of algorithms are shown in the first column, while the second column is the reference of the corresponding algorithm. The last column is the average deviation %. For the J30 instances the deviation is calculated with respect to the optimal solutions, while for J60 and J120 instances the average deviation % is computed with the respect to the CP length. Moreover, the results are obtained on three limits on the number of generated objective evaluations: 1000, 5000, and 50000.

Table 3 Average deviations from the critical path based lower bounds of J120

Algorithm	Reference	Schedules		
		1000	5000	50000
IDCS	**This paper**	**33.43**	**32.69**	**30.48**
ACOSS	Chen *et al*. (2010)	35.19	32.48	30.56
SAILS	Paraskevopoulos *et al*. (2012)	33.32	32.12	30.78
GA	Debels and Vanhoucke (2005)	34.19	32.34	30.82
GA-hybrid FBI	Valls *et al*. (2002)	34.07	32.54	31.24
Special IDCS	**This paper**	**35.94**	**32.91**	**31.52**

SS-FBI	Debels *et al*. (2006)	35.22	33.10	31.57
GA, TS-PR	Kochetov and Stolyar (2003)	34.74	33.36	32.06
GA	Hartmann (2002)	37.19	35.39	33.21
Sampling + BF	Tormos and Lova (2003)	36.24	35.56	34.77
ANGEL	Tseng and Chen (2006)	36.39	34.49	n/a
TS	Nonobe and Ibaraki (2002)	40.86	37.88	35.85

Table 4 Average deviations of the critical path based lower bounds for J60

Algorithm	Reference	Schedules		
		1000	5000	50000
SAILS	Paraskevopoulos *et al*. (2012)	11.05	10.72	10.54
IDCS	**This paper**	**11.78**	**10.99**	**10.67**
ACOSS	Chen *et al*. (2010)	11.72	10.98	10.67
GA	Debels and Vanhoucke (2005)	11.45	10.95	10.68
SS-FBI	Debels *et al*. (2006)	11.73	11.10	10.71
GA-hybrid FBI	Valls *et al*. (2002)	11.56	11.10	10.73
Special IDCS	**This paper**	**11.60**	**11.11**	**10.74**
GA, TS-PR	Kochetov and Stolyar (2003)	11.71	11.17	10.74
GA	Hartmann (2002)	12.21	11.70	11.21
Sampling + BF	Tormos and Lova (2003)	11.88	11.62	11.36
ANGEL	Tseng and Chen (2006)	11.94	11.27	n/a
TS	Nonobe and Ibaraki (2002)	12.97	12.18	11.58

Table 5 Average deviations of the optimal solutions for J30

Algorithm	Reference	Schedules		
		1000	5000	50000
SAILS	Paraskevopoulos *et al*. (2012)	0.03	0.01	0.00
IDCS	**This paper**	**0.09**	**0.04**	**0.01**
GA, TS-PR	Kochetov and Stolyar (2003)	0.10	0.04	0.00
ACOSS	Chen *et al*. (2010)	0.14	0.06	0.01
SS-FBI	Debels *et al*. (2006)	0.27	0.11	0.01
GA	Debels and Vanhoucke (2005)	0.15	0.04	0.02
GA-hybrid FBI	Valls *et al*. (2002)	0.27	0.06	0.02
Special IDCS	**This paper**	**0.31**	**0.06**	**0.02**
TS	Nonobe and Ibaraki (2002)	0.46	0.16	0.05
GA	Hartmann (2002)	0.38	0.22	0.08
Sampling + BF	Tormos and Lova (2003)	0.30	0.17	0.09
ANGEL	Tseng and Chen (2006)	0.22	0.09	n/a
Original DCS	**Bibiks *et al*. (2015)**	**0.44**	**0.25**	**n/a**

For the sake of objectiveness, all the comparisons were made according to the 50,000 performance mode. Table 3, 4, and 5 illustrate that the IDCS could find consistently high quality solutions for all benchmark sets with one exception that SAILS's algorithm performed better than the proposed IDCS in small scale problem or search space, that is J30 and J60. This is the SAIL's technique utilise a scatter search technique in the initialisation process, which can increase the probability in finding solutions in small scale problems; however, if the search space is very huge, our approach performs slightly better , such as the J120 case. The average deviations of solutions obtained with the proposed IDCS for J120, J60 and J30 sets are 30.48%, 10.67% and 0.01% respectively. The results indicate

that the proposed IDCS is capable of finding solutions with higher quality with less iterations. The IDCS shows itself as a competitive algorithm and performs better than or on the same level.

Moreover, as can be furtherly noted, the performance of the IDCS presented in this paper in comparison to its first implementation in Bibiks *et al*. (2015) has been greatly improved by nearly five times. Such level of improvement proves that the use of the EL as a default representation of a solution and mechanisms as event move and event crossover based on this representation is indeed a good choice that brings great benefits for the performance.

Regarding the special version of DCS, from the experimental results it can be seen that the use of big step notation during the generation of the new cuckoo via Lévy flights and subsequent application of event crossover has led to improvement of the performance of the algorithm and consequently acquired better positions in the overall algorithms ranking.

Finally it should be noted that the IDCS demonstrated a high level of a performance comparable to other state-of-the-art metaheuristic methodologies, proving that it is capable of providing a high optimisation rate at low cost. Such performance of the presented algorithm can be explained by several factors. Firstly, the intelligent use of Lévy flights provides a good balance between exploitation and exploration. Secondly, the use of the EL as a default solution representation scheme with the subsequent application of the event move and event crossover operators increases the quality of found solutions as well as makes the process of finding new solutions more efficient. Moreover, such variety of different operators in the DCS ensures that the diversity of a population is kept to a maximum, thus avoiding falling into local optima trap and making it more likely to find global optima.

6 Conclusions

In this paper an Improved Discrete Cuckoo Search (IDCS) has been proposed to solve RCPSPs. It differs from the original CS in several fundamental aspects. Firstly, the event list representation scheme used in the proposed IDCS eliminates core disadvantages of the most commonly used solution representation schemes, such as activity list and random key presentations, where several different solutions can be converted into the same schedule. Secondly, the application of the event list representation introduces the possibility to use such mechanisms as event movement and event recombination. The latter one, namely event recombination, is the novelty of this paper, and in comparison to other recombination mechanisms, is not a pure random nor a context-free operator. It has been designed to combine useful problem-specific information extracted from the parents with the purpose of generating high quality children.

The computational results show that the proposed new version of IDCS is an efficient and high quality algorithm in solving the tested scheduling problems. The IDCS outperforms most of the state-of-the-art algorithms for the RCPSP in this this paper. For practical purposes, it is interesting to note that the solution quality steadily increases with increasing number of generated schedules and the computation time linearly increases with the number of schedules so it is easily predictable.

In the future, further work on the improvement of the DCS will be carried out and the algorithm will be applied in solving more complicated scheduling problems. The probable areas of further applications will include stochastic and multimode RCPSPs. One of the possible areas of improvement is the division of the whole population into several smaller sub-populations. Such approach will divide the search space into regions where each sub-population will be focused on searching for solution within their specified region. This will create an opportunity for a finer search around a local best optima and provide higher chances of reaching global optima. Moreover, this will make the algorithm suitable for application in multimodal scenarios.

7 Acknowledgment

This work is partially funded by the Innovate UK project HARNET – Harmonised Antennas, Radios and Networks under contract no. 100004607.

8 References

Alcaraz, J., & Maroto, C. (2001). A Robust Genetic Algorithm for Resource Allocation in Project Scheduling. *Annals of Operations Research, 102*(1), 83-109.

Bibiks, K., Li, J.-P., Hu, F., & Smith, A. (2015). Discrete Cuckoo Search for Resource-Constrained Project Scheduling Problem. Porto: IEEE Computational Science and Engineering Conference.

Blazewicz, J., Lenstra, J., & Kan, A. (1983). Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics, 5*(1), 11–24.

Boctor, F. F. (1996). Resource-constrained project scheduling by simulated annealing. *Internation Journal of Production Research, 34*(8), 2335-2351.

Bouleimen, K., & Lecocq, H. (2003). A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal of Operational Research, 149*(1), 268–281.

Brucker, P., Knust, S., Schoo, A., & Thiele, O. (1998). A branch and bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research, 107*(2), 272–288.

Chen, B., & Quan, G. (2008). NP-Hard Problems of Learning from Examples. *Fifth International Conference on Fuzzy Systems and Knowledge Discovery.* Shandong.

Chen, W., Shi, Y.-j., Teng, H.-f., Lan, X.-p., & Hu, L.-c. (2010). An efficient hybrid algorithm for resource-constrained project scheduling. *Information Sciences: an International Journal, 180*(6), 1031-1039.

Chetty, S., & Adewumi, A. O. (2013). Comparison Study of Swarm Intelligence Techniques for the Annual Crop Planning Problem. *IEEE Transactions on Evolutionary Computation, 18*(2), 258 - 268.

Debels, D., & Vanhoucke, M. (2005). A Decomposition-Based Genetic Algorithm for the Resource-Constrained Project-Scheduling Problem. *Operations Research, 55*(3), 457 - 469.

Debels, D., De Reyck, B., Leus, R., & Vanhoucke, M. (2006). A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. *European Journal of Operational Research, 169*(2), 638 - 653.

Demeulemeester E. and Herroelen, W. (1992). A Branch-and-Bound Procedure for the Multiple Resource-Constrained Project Scheduling Problem*, Management Science*, 38(12), 1803-1818.

Elazim, S. A., & Ali, E. (2016). Optimal Power System Stabilizers design via Cuckoo Search algorithm. *International Journal of Electrical Power & Energy Systems, 75*(2), 99–107.

Maghsoudlou, H., Afshar-Nadjafia, B., Niaki S.T.A. (2017). Multi-skilled project scheduling with level-dependent rework risk: three multi-objective mechanisms based on cuckoo search, *Applied Soft Computing*, 54, 46-61.

Hartmann, S. (1998). A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics , 45*(7), 733–750.

Hartmann, S. (2002). A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics , 49*(5), 433–448.

Hartmann, S., & Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research, 207*(1), 1–14.

Hartmann, S., & Kolisch, R. (2000). Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling. *European Journal of Operational Research, 127*(2), 394–407.

Herroelen, W., Reyck, B. D., & Demeulemeester, E. (1998). Resource-constrained project scheduling: A survey of recent developments. *Computers & Operations Research, 25*(4), 279–302.

Hsu, C.-C., Blois, Y. d., & Pyle, M. J. (2004). General motors optimizes its scheduling of cold-weather tests. *Interfaces, 34*(5), 334 - 341.

Husbands, P., Jermy, G., McIlhagga, M., & Ives, R. (1996). Two applications of genetic algorithms to component design. *AISB workshop on evolutionary computing*, (pp. 50-61). Chicago, CH.

Kochetov, Y. A., & Stolyar, A. A. (2003). Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem. *Proceedings of the 3rd International Workshop of Computer Science and Information Technologies.* Novosibirks, Russia.

Kolisch, R. (1996). Serial and parallel resource-constrained project scheduling: theory and computation. *European Journal of Operational Research, 90*(2), 320–333.

Kolisch, R., & Hartmann, S. (1999). *Handbook on Recent Advances in Project Scheduling.* Dordrecht: Kluwer.

Kolisch, R., & Hartmann, S. (1999). Heuristic Algorithms for the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis. In *Project Scheduling: Recent models, algorithms and applications* (pp. 147-178). Berlin: Kluwer.

Kolisch, R., & Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research, 174*(1), 23–37.

Kolisch, R., & Padman, R. (2001). An integrated survey of deterministic project scheduling. *Omega, 29*(3), 249–272.

Kolisch, R., & Sprecher, A. (1997). PSPLIB - A project scheduling problem library. *European Journal of Operational Research, 96*(1), 205–216.

Kolisch, R., Sprecher, A., & Drexl, A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science, 41*(10), 1693-1703.

Leccardi, M. (2005). Comparison of three algorithms for Lèvy Noise Generation. *Prroceedings of fifth EUROMECH nonlinear dynamics conference.* Paris.

Mantegna, R. N. (1994). Fast, accurate algorithm for numerical simulation of Lévy stable stochastic processes. *Physical Review E, 49*, 4677-4683.

Tritschler, M., Naber, A., Kolisch, R. (2017) A hybrid metaheuristic for resource-constrained project scheduling with flexible resource profiles, *European Journal of Operational Research*, 262 (1), 262-273.

Merkle, D., Middendorf, M., & Schmeck, H. (2002). Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation, 6*(4), 333-346.

Mingozzi, A., Maniezzo, V., Ricciardelli, S., & Bianco, L. (1998). An Exact Algorithm for the Resource Constrained Project Scheduling Problem Based on a New Mathematical Formulation. *Management Science, 44*(5), 714-729 .

Mori, M., & Tseng, C. C. (1997). A genetic algorithm for multi-mode resource constrained project scheduling problem. *European Journal of Operational Research, 100*(1), 134–141.

Moumene, K., & Ferland, J. A. (2008). New representation to reduce the search space for the resource-constrained project scheduling problem. *RAIRO - Operations Research, 42*(2), 215-228.

Myszkowski, P.B. M., Olech, Ł.P.O., Laszczyk, M., Skowronski, M.E. (2018), Hybrid Differential Evolution and Greedy Algorithm (DEGR) for solving Multi-Skill Resource-Constrained Project Scheduling Problem, *Applied Soft Computing*, 62, 1-14.

Nguyen, T. T., Truong, A. V., & Phung, T. A. (2016). A novel method based on adaptive cuckoo search for optimal network reconfiguration and distributed generation allocation in distribution network. *International Journal of Electrical Power & Energy Systems, 78*(6), 801–815.

Nonobe, K., & Ibaraki, T. (2002). Formulation and tabu search algorithm for the resource constrained project scheduling problem. In C. C. Hansen, *Essays and surveys in metaheuristics* (pp. 557–588). Kluwer Academic Publishers.

Ouaarab, A., Ahiod, B., & Yang, X.-S. (2013). Discrete cuckoo search algorithm for the travelling salesman problem. *Neural Computing and Applications, 24*(7), 1659-1669.

Palpant, M., Artigues, C., & Michelon, P. (2004). LSSPER: Solving the Resource-Constrained Project Scheduling Problem with Large Neighbourhood Search. *Annals of Operations Research, 131*(1),

237-257.
Paraskevopoulos, D., Tarantilis, C., & Ioannou, G. (2012). Solving project scheduling problems with resource constraints via an event list-based evolutionary algorithm. *Expert Systems with Applications, 39*(4), 3983–3994.
Payne, R. B., & Sorenson, M. D. (2005). *The Cuckoos, vol. 15.* Oxford: Oxford University Press.
Sekhar, P., & Mohanty, S. (2016). An enhanced cuckoo search algorithm based contingency constrained economic load dispatch for security enhancement. *International Journal of Electrical Power & Energy Systems, 75*(2), 303–310.
Sprecher, A. (2000). Scheduling Resource-Constrained Projects Competitively at Modest Memory Requirements. *Management Science, 46*(5), 710-723.
Sprecher, A., Kolisch, R., & Drexl, A. (1995). Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. *European Journal of Operational Research, 80*(1), 94–102.
Teymourian, E., Kayvanfar, V., Komaki, G., & Zandieh, M. (2016). Enhanced intelligent water drops and cuckoo search algorithms for solving the capacitated vehicle routing problem. *Information Sciences, 334*(6), 354–378.
Tormos, P., & Lova, A. (2001). A Competitive Heuristic Solution Technique for Resource-Constrained Project Scheduling. *Annals of Operations Research, 102*(1), 65-81.
Tormos, P., & Lova, A. (2003). An efficient multi-pass heuristic for project scheduling with constrained resources. *Journal of Production Research, 41*, 1071–1086.
Tseng, L., & Chen, S. (2006). A hybrid metaheuristic for the resource-constrained project scheduling problem. *European Journal of Operational Research, 175*(2), 707-721.
Valian , E., S. Tavakoli, S. Mohanna, (2013) A. Haghi, Improved cuckoo search for reliability optimization problems, Comput. Ind. Eng. 64, 459–468.
Valls, V., Ballestin, F., & Quintanilla, S. (2002). A hybrid genetic algorithm for the RCPSP with the peak crossover operator. *The Eighth International Workshop on Project Management and Scheduling.* Madrid.
Valls, V., Ballestín, F., & Quintanilla, S. (2005). Justification and RCPSP: A technique that pays. *European Journal of Operational Research, 165*(2), 375–386.
Yang, X.-S., & Deb, S. (2009). Cuckoo Search via Lévy flights. *World Congress on Nature & Biologically Inspired Computing.* Coimbatore.
Yang, X.-S., & Deb, S. (2010). Engineering optimisation by cuckoo search. *International Journal of Mathematical Modelling and Numerical Optimisation, 1*(4), 330 -343.
Yang, X.-S. (2013). Cuckoo Search and Firefly Algorithm: Theory and Applications, *Springer*, 2013.
Zhou, X., & Zhong, M. (2007). Single-track train timetabling with guaranteed optimality: Branch-and-bound algorithms with enhanced lower bounds. *Transportation Research Part B: Methodological, 41*(3), 320–341.
Zhou, Y., & Chen, Y. (2002). Business process assignment optimization. *IEEE International Conference on Systems, Man and Cybernetics.* Paris.
Ziarati, K., Akbari, R., Zeighami, V. (2011). On the performance of bee algorithms for resource-constrained project scheduling problem, Applied Soft Computing, 11 (4) 3720-3733.